Lifting artificial restrictions on universal character names

Document #:	P2620R0
Date:	2022-07-09
Programming Language C++	
Audience:	EWG, SG-22

Reply-to: Corentin Jabot <corentin.jabot@gmail.com>

Abstract

We propose to lift restrictions on *universal-character-names* in identifiers.

Motivation

There are restrictions on the constitution of *universal-character-names* that seem artificial, and we should lift them[1]!

```
\N{LATIN CAPITAL LETTER I} = 42; // ERROR: I is in the basic character set
\N{LATIN CAPITAL LETTER I WITH DOT ABOVE} = 42 ;// Ok
```

This is by no mean a major issue in C++, as we don't put restrictions on *universal-character-names* in string literals (unlike C), but it is somewhat inconsistent with the lexing model.

Instead of restricting *universal-character-names* values, we can instead mandate that they are part of valid identifiers outside of strings.

Comparison With C

C does not allow *universal-character-names* to designate elements of the basic character set:

> 2 A universal character name shall not designate a codepoint where the hexadecimal value is: - less than 00A0 other than 0024 ($), 0040 (@), or 0060 (‘);

This has been a pain point for users who would like to consistently use `\u` in string literals as part of code generation processes.

- LLVM issue: Unicode string literals
- Why C99 has such an odd restriction for universal character names?
- Restrictions to Unicode escape sequences in C11

[1] This is a small cleanup that isn't worth doing unless we can spend very little time on it, classified as low priority bucket 72.

I hope that both languages regain consistency by:

- Not restricting UCNs in string literals
- Not putting restrictions on UCNs in identifiers beyond what naturally falls out of the grammar of identifiers.

Wording

[lex.separate] Separate translation

4. The source file is decomposed into preprocessing tokens and sequences of whitespace characters (including comments). A source file shall not end in a partial preprocessing token or in a partial comment. Each comment is replaced by one space character. New-line characters are retained. Whether each nonempty sequence of whitespace characters other than new-line is retained or replaced by one space character is unspecified. As characters from the source file are consumed to form the next preprocessing token (i.e., not being consumed as part of a comment or other forms of whitespace), except when matching a *c-char-sequence*, *s-char-sequence*, *r-char-sequence*, *h-char-sequence*, or *q-char-sequence*, *universal-character-name* s are recognized and replaced by the designated element of the translation character set. If a *universal-character-name* was replaced during the formation of a preprocessing token, that token shall match the syntax of an *identifier*. The process of dividing a source file's characters into preprocessing tokens is context-dependent. [*Example:* See the handling of < within a `#include` preprocessing directive. — *end example*]

[lex.charset] Character sets

A *universal-character-name* designates the character in the translation character set whose UCS scalar value is the hexadecimal number represented by the sequence of *hexadecimal-digit* s in the *universal-character-name*. The program is ill-formed if that number is not a UCS scalar value. ~~If a *universal-character-name* outside the *c-char-sequence*, *s-char-sequence*, or *r-char-sequence* of a *character-literal* or *string-literal* (in either case, including within a *user-defined-literal*) corresponds to a control character or to a character in the basic character set, the program is ill-formed.~~ [*Note:* A sequence of characters resembling a *universal-character-name* in an *r-char-sequence* does not form a *universal-character-name*. — *end note*]